## SQL JOIN

A JOIN clause is used to combine rows from same, two, or more tables, based on a related column between them.

**1. Inner Join:**

An Inner join retrieves rows from both tables that satisfy the join condition. It combines rows from two tables based on a related column between them.



**Note:** The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns.

### INNER JOIN Syntax

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2  
ON table1.column\_name *=* table2.column\_name*;*

Example: Suppose we have another table called "Department" with columns "Major" and "DepartmentName". We want to join the "Student" table with the "Department" table based on the "Major" column to get the department names for each student.

CREATE TABLE Department (

Major VARCHAR(50) PRIMARY KEY,

DepartmentName VARCHAR(50)

);

INSERT INTO Department (Major, DepartmentName)

VALUES

('Computer Science', 'Engineering'),

('Mathematics', 'Science'),

('Physics', 'Science'),

('Chemistry', 'Science'),

('Biology', 'Science'),

('English', 'Humanities');

-- Inner join to get department names for each student

SELECT s.FirstName, s.LastName, s.Major, d.DepartmentName

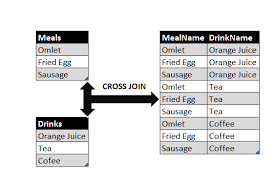
FROM Student s

INNER JOIN Department d

ON s.Major = d.Major;

**2. Cross Join:**

A Cross join (Cartesian join) returns the Cartesian product of the two tables, meaning it combines each row of the first table with every row of the second table.



### CROSS JOIN Syntax

SELECT column\_name(s)  
FROM table1  
CROSS JOIN table2*;*

Example: Suppose we want to generate all possible combinations of students and departments.

-- Cross join to generate all possible combinations of students and departments

SELECT s.FirstName, s.LastName, s.Major, d.DepartmentName

FROM Student s

CROSS JOIN Department d;

**3. Outer Join:**

An Outer join retrieves all rows from one table and only those rows from the other table that satisfy the join condition. It includes unmatched rows from one or both tables depending on the type of Outer join (Left, Right, or Full).

## LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.



### LEFT JOIN Syntax

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2ON table1.column\_name *=* table2.column\_name*;*

Example: Let's perform a Left Outer join to get all students and their corresponding departments, including students without departments.

-- Left Outer join to get all students and their corresponding departments

SELECT s.FirstName, s.LastName, s.Major, d.DepartmentName

FROM Student s

LEFT OUTER JOIN Department d

ON s.Major = d.Major;

## RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.



### RIGHT JOIN Syntax

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2ON table1.column\_name *=* table2.column\_name*;*

Example: Right Outer join to get all departments and their corresponding students, including departments without students

SELECT s.FirstName, s.LastName, s.Major, d.DepartmentName

FROM Student s

RIGHT OUTER JOIN Department d

ON s.Major = d.Major;

## FULL OUTER JOIN

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.



### FULL OUTER JOIN Syntax

SELECT column\_name(s)FROM table1  
FULL OUTER JOIN table2ON table1.column\_name *=* table2.column\_nameWHERE condition*;*

Example: Full Outer join to get all students and their corresponding departments, including unmatched rows from both tables

SELECT s.FirstName, s.LastName, s.Major, d.DepartmentName

FROM Student s

FULL OUTER JOIN Department d

ON s.Major = d.Major;

In some SQL Server versions, there isn't a native FULL OUTER JOIN syntax using FULL OUTER JOIN. Instead, you typically emulate a full outer join using a combination of a left outer join and a right outer join, along with UNION to combine the results. Here's an example of how you can perform a full outer join between the "Student" and "Department" tables:

SELECT s.FirstName, s.LastName, s.Major, d.DepartmentName

FROM Student s

LEFT OUTER JOIN Department d ON s.Major = d.Major

UNION

SELECT s.FirstName, s.LastName, s.Major, d.DepartmentName

FROM Student s

RIGHT OUTER JOIN Department d ON s.Major = d.Major

WHERE s.Major IS NULL;

**4. Self Join:**

A Self join is a join that joins a table with itself. It is used to combine rows with other rows in the same table based on related columns.

### SELF JOIN Syntax

SELECT column\_name(s)  
FROM table1 T1, table1 T2  
WHERE condition*;*

Example: Suppose we want to find all pairs of students who have the same major.

SELECT s1.FirstName AS Student1FirstName, s1.LastName AS Student1LastName, s2.FirstName AS Student2FirstName, s2.LastName AS Student2LastName, s1.Major

FROM Student s1, Student s2

WHERE s1.Major = s2.Major AND s1.StudentID != s2.StudentID;

Or:

SELECT s1.FirstName AS Student1FirstName, s1.LastName AS Student1LastName, s2.FirstName AS Student2FirstName, s2.LastName AS Student2LastName, s1.Major

FROM Student s1

INNER JOIN Student s2

ON s1.Major = s2.Major AND s1.StudentID != s2.StudentID;

More examples:

We create and add two other tables (with few records) to our database. In these tables:

* The "Course" table stores information about different courses offered by departments. Each course is associated with a specific department.
* The "Enrollment" table stores information about student enrollments in courses. Each enrollment links a student to a course, along with their grade in that course.

These tables create a simple relational schema representing student enrollment in courses, with foreign key relationships established between the "Course" table and the "Department" table, and between the "Enrollment" table and the "Student" and "Course" tables.

CREATE TABLE Course (

CourseID INT PRIMARY KEY,

CourseName VARCHAR(50),

Major VARCHAR(50),

FOREIGN KEY (Major) REFERENCES Department(Major)

);

INSERT INTO Course (CourseID, CourseName, Major) VALUES

(1, 'Introduction to Computer Science', 'Computer Science'),

(2, 'Calculus I', 'Mathematics'),

(3, 'General Physics', 'Physics'),

(4, 'Organic Chemistry', 'Chemistry'),

(5, 'Literature and Composition', 'English'),

(6, 'Data Structures and Algorithms', 'Computer Science'),

(7, 'Linear Algebra', 'Mathematics'),

(8, 'Electricity and Magnetism', 'Physics'),

(9, 'Biochemistry', 'Chemistry'),

(10, 'American Literature', 'English'),

(11, 'Advanced Programming', 'Computer Science'),

(12, 'Differential Equations', 'Mathematics'),

(13, 'Modern Physics', 'Physics'),

(14, 'Inorganic Chemistry', 'Chemistry'),

(15, 'Shakespearean Studies', 'English');

CREATE TABLE Enrollment (

EnrollmentID INT PRIMARY KEY,

StudentID INT,

CourseID INT,

Grade DECIMAL(5,2),

FOREIGN KEY (StudentID) REFERENCES Student(StudentID),

FOREIGN KEY (CourseID) REFERENCES Course(CourseID));

INSERT INTO Enrollment (EnrollmentID, StudentID, CourseID, Grade) VALUES

(1, 1, 1, 90.5),

(2, 2, 2, 85.0),

(3, 3, 3, 92.5),

(4, 4, 4, 88.0),

(5, 5, 5, 95.5),

(6, 6, 6, 91.0),

(7, 7, 7, 89.5),

(8, 8, 8, 93.0),

(9, 9, 9, 90.5),

(10, 10, 10, 94.0),

(11, 11, 11, 87.5),

(12, 12, 12, 84.0),

(13, 13, 13, 91.5),

(14, 14, 14, 88.5),

(15, 15, 15, 96.0);

**Inner Join:**

Retrieve student information along with their enrolled courses and grades:

SELECT s.FirstName, s.LastName, c.CourseName, e.Grade FROM Student s INNER JOIN Enrollment e ON s.StudentID = e.StudentID INNER JOIN Course c ON e.CourseID = c.CourseID;

**Left Join:**

Get all students along with their enrollment information (including those who are not enrolled):

SELECT s.FirstName, s.LastName, e.CourseID, e.Grade FROM Student s LEFT JOIN Enrollment e ON s.StudentID = e.StudentID;

**Right Join:**

Get all students along with their enrollment information (including those who are not enrolled):

SELECT s.FirstName, s.LastName, e.CourseID, e.Grade FROM Student s RIGHT JOIN Enrollment e ON s.StudentID = e.StudentID;

**Full Join:**

Get all students along with their enrollment information (including those who are not enrolled):

SELECT s.FirstName, s.LastName, e.CourseID, e.Grade FROM Student s FULL JOIN Enrollment e ON s.StudentID = e.StudentID;

**Cross Join:**

Get all combinations of students and courses:

SELECT s.FirstName, s.LastName, c.CourseName FROM Student s CROSS JOIN Course c;

**Data integrity**

**Data integrity** is the maintenance of, and the assurance of, data accuracy and consistency over its entire life-cycle.It is a critical aspect to the design, implementation, and usage of any system that stores, processes, or retrieves data.

**Data integrity** is the opposite of data corruption. The overall intent of any data integrity technique is the same: ensure data is recorded exactly as intended. Moreover, upon later retrieval, ensure the data is the same as when it was originally recorded. In short, data integrity aims to prevent unintentional changes to information. Data integrity is not to be confused with data security, the discipline of protecting data from unauthorized parties.

Any unintended changes to data as the result of a storage, retrieval or processing operation, including malicious intent, unexpected hardware failure, and human error, is failure of data integrity. If the changes are the result of unauthorized access, it may also be a failure of data security.

Data integrity contains guidelines for data retention, specifying or guaranteeing the length of time data can be retained in a particular database (typically a relational database). To achieve data integrity, these rules are consistently and routinely applied to all data entering the system, and any relaxation of enforcement could cause errors in the data. Implementing checks on the data as close as possible to the source of input (such as human data entry), causes less erroneous data to enter the system. Strict enforcement of data integrity rules results in lower error rates, and time saved troubleshooting and tracing erroneous data and the errors it causes to algorithms.

**Types of integrity constraints**

Data integrity is normally enforced in a database system by a series of integrity constraints or rules. Three types of integrity constraints are an inherent part of the relational data model: entity integrity, referential integrity and domain integrity.

* **Entity integrity** concerns the concept of a primary key. Entity integrity is an integrity rule which states that every table must have a primary key and that the column or columns chosen to be the primary key should be unique and not null.
* **Referential integrity** concerns the concept of a foreign key. The referential integrity rule states that any foreign-key value can only be in one of two states. The usual state of affairs is that the foreign-key value refers to a primary key value of some table in the database. Occasionally, and this will depend on the rules of the data owner, a foreign-key value can be null. In this case, we are explicitly saying that either there is no relationship between the objects represented in the database or that this relationship is unknown.
* **Domain integrity** specifies that all columns in a relational database must be declared upon a defined domain. The primary unit of data in the relational data model is the data item. Such data items are said to be non-decomposable or atomic. A domain is a set of values of the same type. Domains are therefore pools of values from which actual values appearing in the columns of a table are drawn.
* **User-defined integrity** refers to a set of rules specified by a user, which do not belong to the entity, domain and referential integrity categories.

Having a well-controlled, and well-defined data-integrity system increases:

* stability (one centralized system performs all data integrity operations)
* performance (all data integrity operations are performed in the same tier as the consistency model)
* re-usability (all applications benefit from a single centralized data integrity system)
* maintainability (one centralized system for all data integrity administration).

**UNION Operator**

The UNION operator is used to combine the result-set of two or more select statements.

* Every SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in every SELECT statement must also be in the same order

UNION Syntax

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.

UNION ALL Syntax

SELECT column\_name(s) FROM table1  
UNION ALL  
SELECT column\_name(s) FROM table2;

**Note:** The column names in the result-set are usually equal to the column names in the first SELECT statement.

UNION selects only distinct values. Use UNION ALL to also select duplicate values.

# **INTERSECT Operator**

The INTERSECT operator in SQL is used to combine two SELECT statements but the dataset returned by the INTERSECT statement will be the intersection of the data sets of the two SELECT statements. In simple words, the INTERSECT statement will return only those rows which will be common to both of the SELECT statements.



**INTERSECT Syntax**

SELECT column1 , column2 ….

FROM table\_names

WHERE condition

INTERSECT

SELECT column1 , column2 ….

FROM table\_names

WHERE condition

## EXCEPT Operator

The EXCEPT operator in SQL is used to retrieve all the unique records from the left query, except the records that are present in the result set of the right query.

In other words, this operator compares the distinct values of the left query with the result set of the right query. If a value from the left query is found in the result set of the right query, it is excluded from the final result.

**EXCEPT Syntax**

SELECT column1, column2,...

FROM table1, table2,...,

[Conditions] //optional

EXCEPT

SELECT column1, column2,...

FROM table1, table2,...

[Conditions] //optional

**Example:**

UNION

1. Combine the list of student first names and last names:

SELECT FirstName FROM Student UNION SELECT LastName FROM Student;

1. Get the distinct list of majors from the student and department tables:

SELECT Major FROM Student UNION SELECT Major FROM Department;

UNION ALL

1. Combine the list of student first names and last names, including duplicates:

SELECT FirstName FROM Student UNION ALL SELECT LastName FROM Student;

1. Get all majors from the student and department tables, including duplicates:

SELECT Major FROM Student UNION ALL SELECT Major FROM Department;

INTERSECT

1. Get the list of majors that are common between students and departments:

SELECT Major FROM Student INTERSECT SELECT Major FROM Department;

1. Find the courses that are common between the course and enrollment tables:

SELECT CourseID FROM Course INTERSECT SELECT CourseID FROM Enrollment;

EXCEPT

1. Find the majors that are unique to the student table (not found in the department table):

SELECT Major FROM Student EXCEPT SELECT Major FROM Department;

1. Retrieve the courses that are unique to the course table (not found in the enrollment table):

SELECT CourseID FROM Course EXCEPT SELECT CourseID FROM Enrollment;